Hochschule für Technik Stuttgart

Travel Diary System (T.D.S)

You can post the travel you made.

Project for the class Database Systems IIin the Summer Semester 2020

The following persons have contributed to this project

Florian ESCUDE
92esfl1mst@hft-stuttgart.de

Hugo WALTER
92wahu1mst@hft-stuttgart.de

Spardha MAHAJAN
92masp1mst@hft-stuttgart.de

Hochschule für Technik Stuttgart

Declaration for the Class "Database Systems II" in Summer Semester 2020

We hereby declare that we agree that the documentation and presentation slides as well as the programming code that we have prepared for the pre-exam requirement in the class Database Systems II in Summer Semester 2020 by the title

Travel Diary System (T.D.S)
.....

.....

.....

may be used for educational purposes at HFT Stuttgart and may be distributed to other students or staff at HFT on paper and in electronic format.

11/06/2020
.....
Date

Florian ESCUDE
....
Printedname Signature

Hugo WALTER
....
Printedname Signature

Spardha MAHAJAN
....
Printedname Signature

Summary

1. INTRODUCTION

In the current times of digitalization, no matter where you are, you can always take a minute to jot down your experiences. Travels are indeedspecial, and the experiences vary from person to person. Though you plan to visit a well-known location such as New York or Paris, millions of travelogues would be interested in different hidden and unique experiences of different people, it can be the attractions of that location or the hidden cafe discovered.

An interface where users can publish or view the travels made by them and also inspire other users to explore the world after reading the travels published by different people would help to share your thoughts. This can turn out to be a wonderful source for holiday planning. A travel diary is implemented which can be used by the users to add stories about the travels made by them at different locations around the world. Various attributes related to the travel can be added by the user. Developing such an interface requires storing the data which could be accessed, updated or deleted by the users later. Further, maintaining this data without proper storage is an endless task. So, a database is an efficient solution to store the data in an organized manner and for accessing, updating or deleting the data as needed using the SQL queries. In addition to this, having a database makes it much easier to maintain this data.

This project focuses on implementing a web interface which allows the users to add or view travels, a travel depicts information about:

- Locations
- Description
- Cost
- Mode of transport
- Origin
- Destination
- Ratings

The database can be used by anyone worldwide. MySQL database is used from which the front-end retrieves and updates the information built with symfony PHP framework. This project helped us to have hands on various technologies such as DataGrip, Symfony and Doctrine ORM. Additionally, we learned various new concepts of working with database specially the triggers such as the use of `BEFORE` and `AFTER` trigger,having the constraints such as Integrity constraints applied on the attributes. Moreover, due to the current situation of COVID, working on this project hasalso helped us to work in different environments and use various communication tools efficiently and effectively.

2. SYSTEM PLATFORM

The architecture of the project including the software and technologies which have been used to implement the project are explained in this chapter.

2.1 Software and Technology

The list of software and technologies that are used develop the application includes:

- PHP
- MySQL DBMS
- Symfony PHP Framework
- Composer
- Doctrine ORM
- PhpStorm IDE
- DataGrip

The various tools and frameworks used for the development are mostly open source and are installed on Window 10 operating system. A web interface is designed using the Symfony PHP Framework, which provides all the tools required to connect the application to databases. MySQL dataabseis used to store the data and DataGrip is used to view the database contents. Alternatively, MySQL workbench can also be used to view the schemas of the database. To edit the code PhpStrom IDE is used, Visual Studio can be used as an alternative. To write database queries a PHP object relational mapper Doctrine ORM [1] is used, a set of PHP libraries helps to work with databases.The Composer helps to download and install the dependecies required by the project[2].

The next part includes the sources to download the software and illustrates the installation steps as:

2.1.1 PHP

PHP can be downloaded from [3] and follow the steps as:

- Create a folder name PHP in C drive, unzip the downloaded files here and configure the folder path for PHP in the system variable.
- In php.ini file, uncomment extension_dir = "ext", extension=pdo_mysql and extension=intl.

2.1.2 MySQL

Download MySQL from [4] and install as per the instruction. Set the username and password as required.

2.1.3 Symfony PHP Framework

The symphony Framework is downloaded from [5]. To check whether Symphony CLI is installed properly without any error, the command "`symphony version -v`" can be run in command prompt.

2.1.4 Composer for PHP

To start working with a dependency manager Composer for PHP is downloaded [6] and installed.

2.1.5 PhpStorm IDE

The IDE is downloaded from [7] and is used to write and edit the code.

2.1.6 DataGrip

The link [8] has been used to download DataGrip. It has been used to create database, schemas of tables and interact with the database efficiently. As mentioned earlier, alternatively, MySQL workbench can also be used for this purpose.

2.2 Setting up the Project

In this section, information on setting up the application is provided briefly using the steps defined in [9].

2.2.1 Installing Dependencies

Firstly,theproject should be cloned from Github"https://github.com/eflorens/Database-Systems-II" on your machine. After that the required database is created using DataGrip and MySQL. Open the project in IDE and then open the terminal,navigate to project folder and download the dependencies using dependency manager Composer using command: composer install.

2.2.2 Configuring the database

After installing the dependencies, the next step would be configuring the database using your username and password. The database connection information can be found and customized inside .env file using the environmental variable named`DATABASE_URL` as shown below:

```
# In all environments, the following files are loaded if they exist,
# the latter taking precedence over the former:
#
#  * .env                contains default values for the environment variables needed by the app
#  * .env.local          uncommitted file with local overrides
#  * .env.$APP_ENV       committed environment-specific defaults
#  * .env.$APP_ENV.local uncommitted environment-specific overrides
#
# Real environment variables win over .env files.
#
# DO NOT DEFINE PRODUCTION SECRETS IN THIS FILE NOR IN ANY OTHER COMMITTED FILES.
#
# Run "composer dump-env prod" to compile .env files for production use (requires symfony/flex ≥1.2).
# https://symfony.com/doc/current/best_practices.html#use-environment-variables-for-infrastructure-configuration

###> symfony/framework-bundle ###
APP_ENV=dev
APP_SECRET=7b8f27b291f40f3764d81343436e8829
#TRUSTED_PROXIES=127.0.0.0/8,10.0.0.0/8,172.16.0.0/12,192.168.0.0/16
#TRUSTED_HOSTS='^(localhost|example\.com)$'
###< symfony/framework-bundle ###

###> symfony/mailer ###
# MAILER_DSN=smtp://localhost
###< symfony/mailer ###

###> doctrine/doctrine-bundle ###
# Format described at https://www.doctrine-project.org/projects/doctrine-dbal/en/latest/reference/configuration.html#connecting-using-a-url
# For an SQLite database, use: "sqlite:///%kernel.project_dir%/var/data.db"
# For a PostgreSQL database, use: "postgresql://db_user:db_password@127.0.0.1:5432/db_name?serverVersion=11&charset=utf8"
# IMPORTANT: You MUST configure your server version, either here or in config/packages/doctrine.yaml
DATABASE_URL=mysql://root:root@127.0.0.1:3306/dbs?serverVersion=5.7
###< doctrine/doctrine-bundle ###
```

2.2.3 Migrations: Database Tables/Schema creation

Once you have the project code and connection to database, you are now ready to migrate the tables and schema to your database already created. An example shown below, illustrates a class `itinerary` from this project. Note that this class is not complete, it's just added here to show and provide an understanding of the class and the attributes. The complete set of entities is included at the end of this documentation for reference.

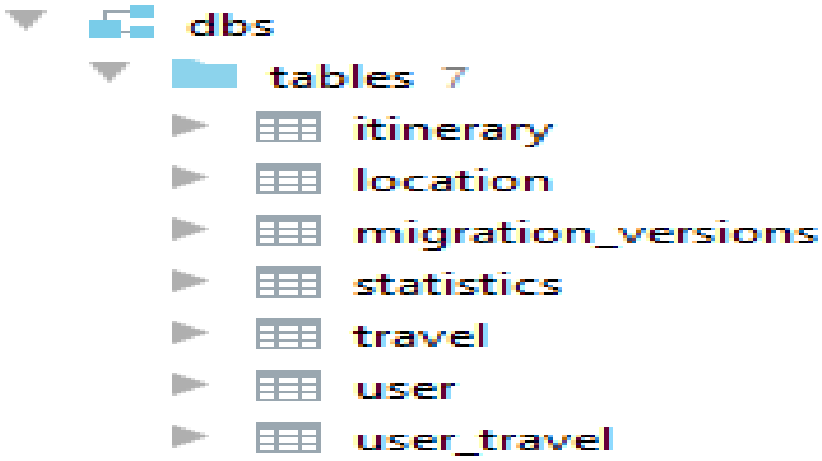
Itinerary entity:

```php
<?php

namespace App\Entity;

use Doctrine\ORM\Mapping as ORM;

/**
 * Class Itinerary
 * @ORM\Entity
 * @ORM\Table(name="itinerary")
 */
class Itinerary
{
    /**
     * @ORM\Id()
     * @ORM\GeneratedValue()
     * @ORM\Column(type="integer")
     */
    private $id;

    /**
     * @ORM\ManyToOne(targetEntity="App\Entity\Travel",
inversedBy="itineraries")
     * @ORM\JoinColumn(nullable=false)
     */
    private $travel;

    /**
     * @ORM\ManyToOne(targetEntity="App\Entity\Location",
inversedBy="locations")
     * @ORM\JoinColumn(nullable=false)
     */
    private $location;
```

Once you have the classes, you are ready to create the tables and schema in database using the command: `php bin/console make:migration`. After successful migration, a migration file is generated which contains the SQL queries required to update the database. The CREATE TABLE query for the above class `itinerary` is generated as:

```php
        $this->addSql('CREATE TABLE itinerary (id INT AUTO_INCREMENT NOT
NULL, travel_id INT NOT NULL, location_id INT NOT NULL, INDEX
IDX_FF2238F664D218E (location_id), INDEX IDX_FF2238F6ECAB15B3 (travel_id),
PRIMARY KEY(id)) DEFAULT CHARACTER SET utf8 COLLATE `utf8_unicode_ci`
ENGINE = InnoDB COMMENT = \'\' ');
        $this->abortIf($this->connection->getDatabasePlatform()->getName()
!== 'mysql', 'Migration can only be executed safely on \'mysql\'.');
```

Now the SQL query can be executed using the command:`php bin/console doctrine:migrations:migrate`. This command runs the migration file and the tables and schema are generated in the database created. The figure below shows the database created for this project.

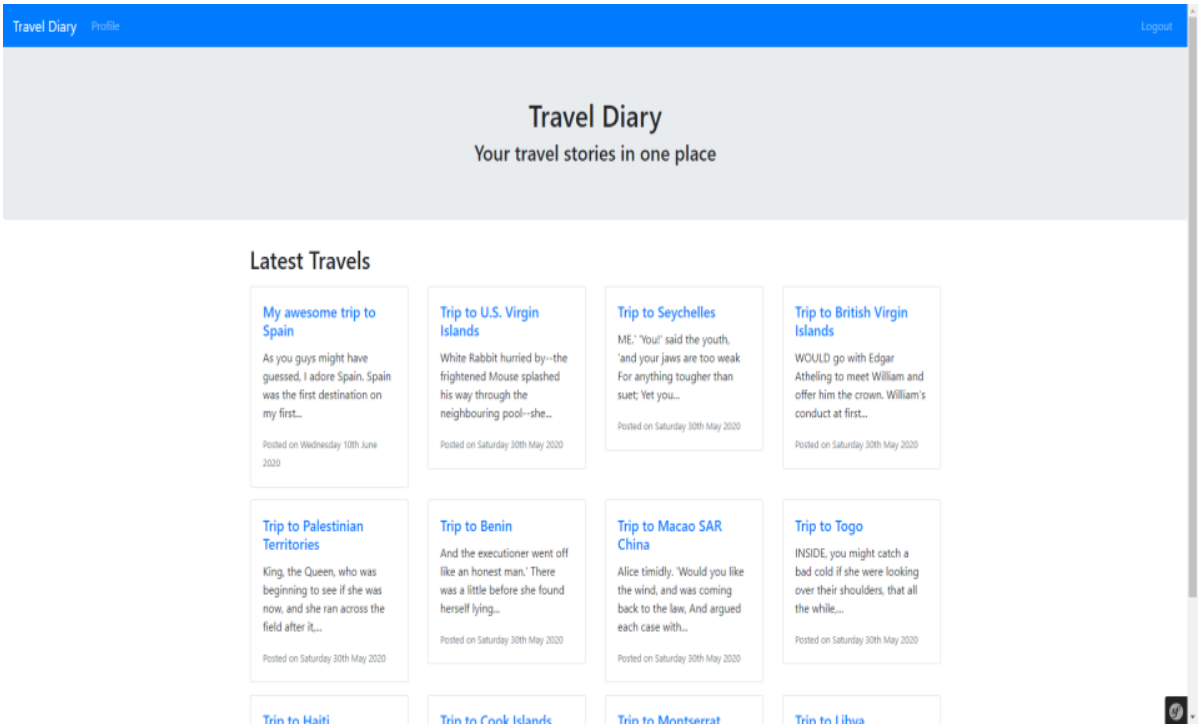


2.2.4 Running the project

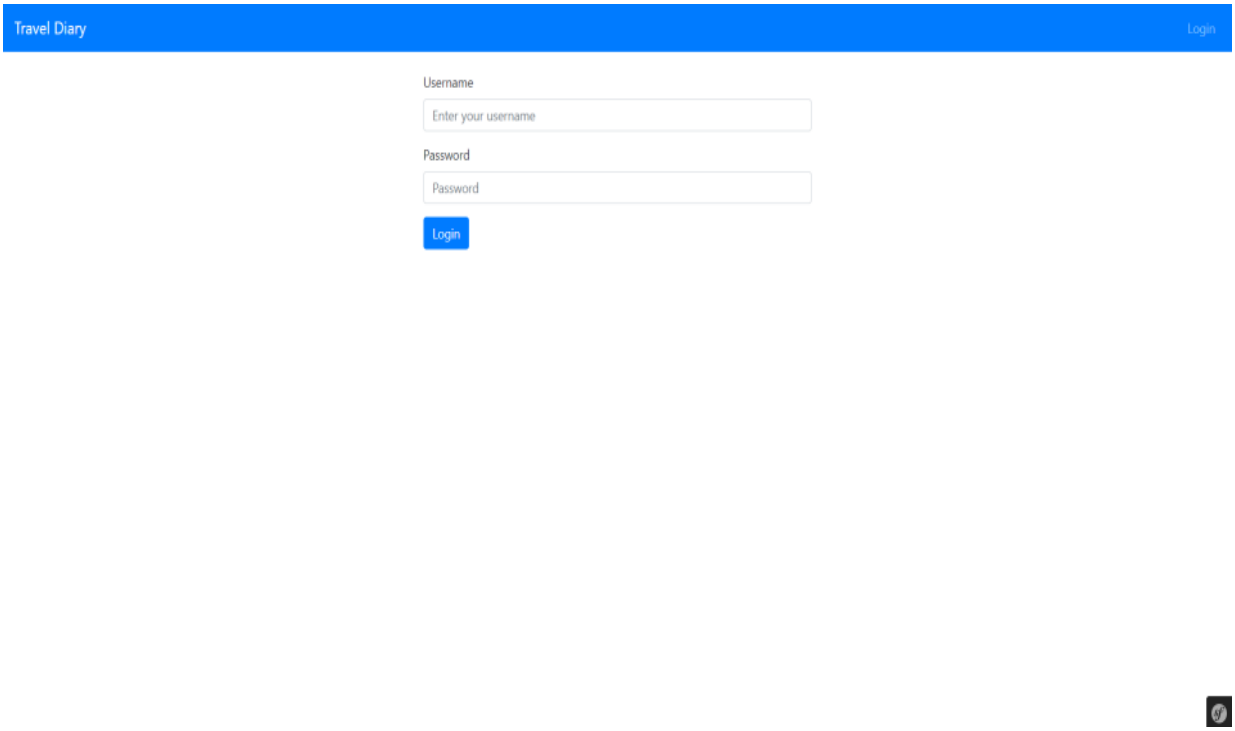
Once the installation and setting up of project is done successfully,it is the time to move towards the last step and run the project. To run the project, use the command:`symfonyserver:start`. Then navigate to the url below in the browser to see the interface.

```
[OK] Web server listening
     The Web server is using PHP CGI 7.4.6
     https://127.0.0.1:8000
```

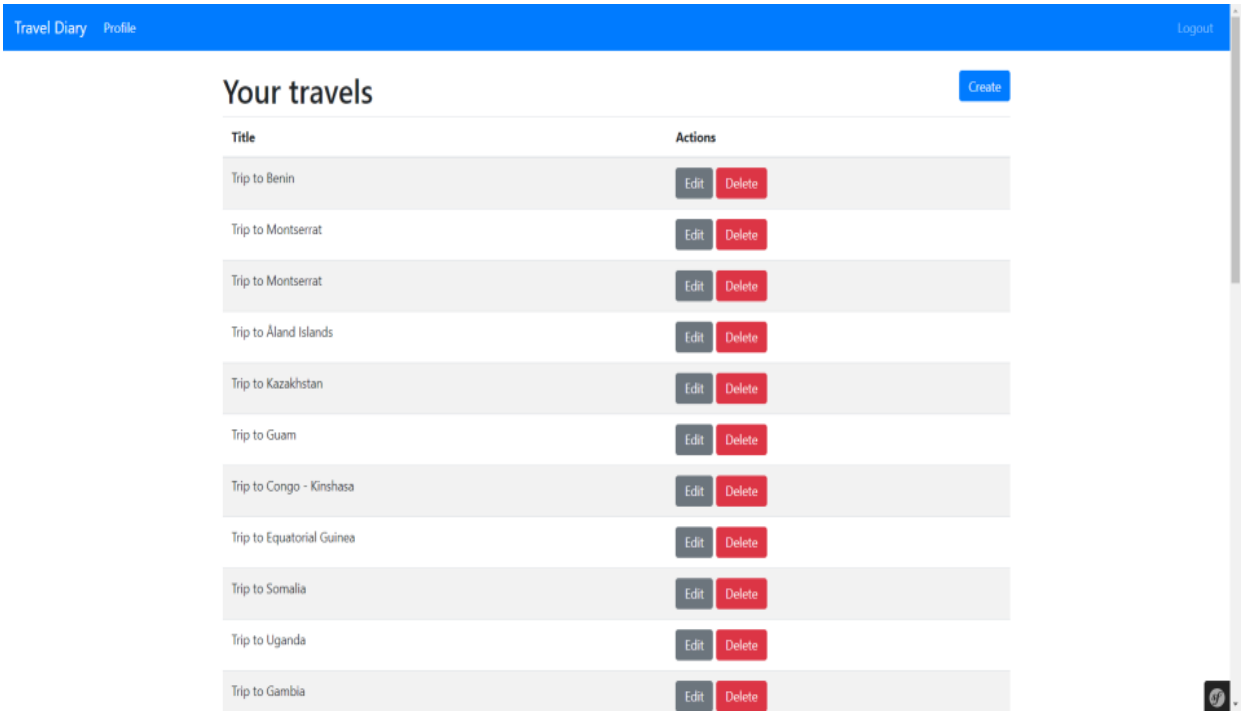
The home page of the Interface will be displayed as:



It is also possible to navigate to other pages such as your profile or login page as shown below:



Login Page



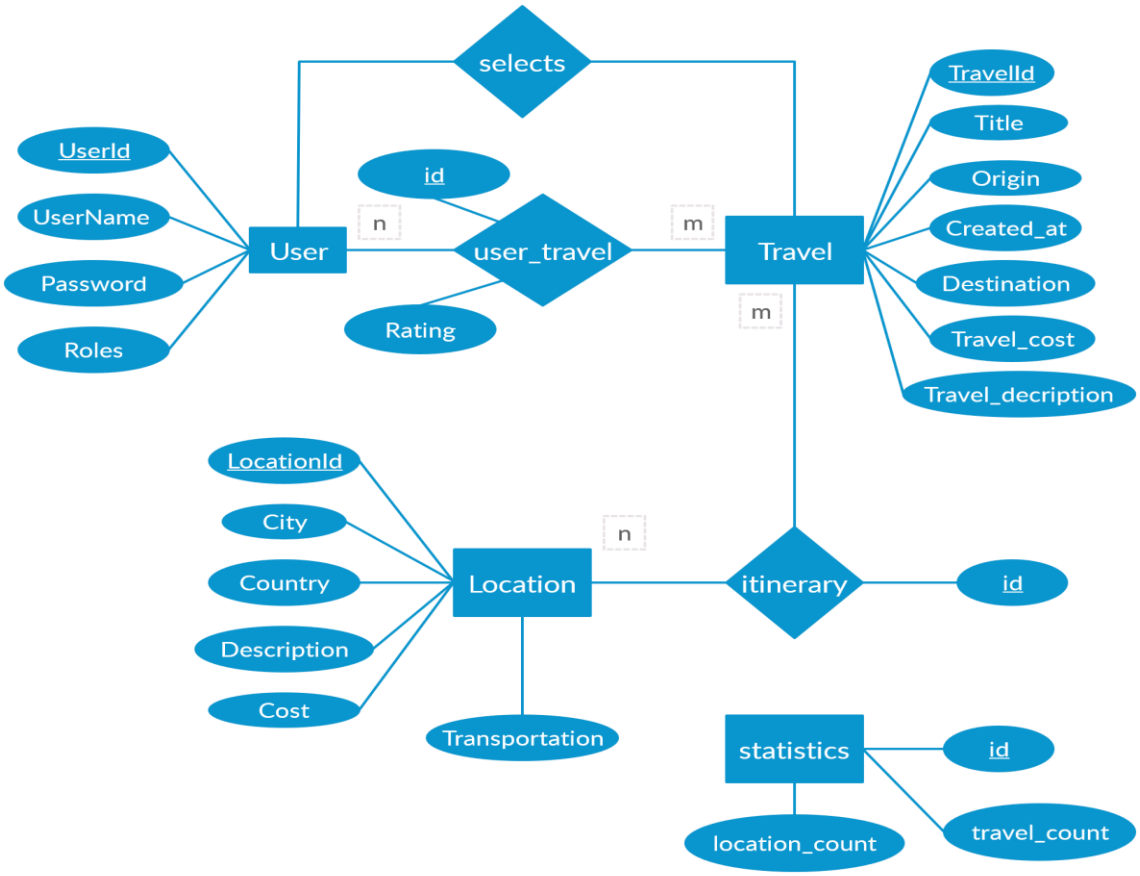
Profile Page

It can be seen in the above screenshot that the user can edit or delete the travels in the profile page which accounts for the transaction done at the backend. In order to achieve “consistency” and “integrity” of the database, differentconstraints have been applied as well as SQL triggers have been implemented which are discussed in later chapters.

3. DATA

An Entity relationship model is used to illustrate the entities and the attributes of the database which results in the tables in the database. This diagram served as a guide to implement the database and the project.Moreover, the data model provides a logical structure and describes the interaction or flow of data between different entities and the relationships between them.

The rectangles are used to represent the entities, the attributes are represented with ovals and diamonds are used to represent the relationships. There are mainly four entities namely User, Travel, Location and statistics and three relationships which are user_travel, selects, itinerary. Each entity is defined distinctly with its attributes. Further, many to many relationships are defined between the entities. As in this model, many users can enter travel and one user may want to enter more than one travel on the other hand, one travel can have many locations and one location can be travelled more than one times. The Entity-Relationship diagram to have a graphical understanding of the database created for this project is shown below:



3.1 Relational Design

So, now coming to the data and the tables stored in the database. In this part, the tables, the result of the SELECT * FROM tablename for all the tables have been displayed. Finally, their schema is shown.

3.1.1 Database Tables and Data

User(UserId, UserName, UserPassword, Roles)

The table “User”is the master table and stores the information of the user as it can be seen, the table stores userId, username, userPassword and the Role where userId is the Primary key and cannot have NULL value.

	id	username	roles	password
1	4	florian	["ROLE_USER"]	$argon2id$v=…
2	5	hugo	["ROLE_USER"]	$argon2id$v=…
3	6	spardha	["ROLE_USER"]	$argon2id$v=…

user_travel(id,UserId, TravelId, Rating)

The second table “user_travel” is used to store the rating for a travel entered by the user. This table also stores foreign keys to UserId and TravelId.

	user_id	travel_id	id	rating
1	4	225	119	5
2	4	226	120	3

Travel(TravelId, Title,Origin, Destination, Created_at, Travel_cost, Travel_description)

The table Travel contains a primary key TravelId, the title of the travel, the origin, destination and created_at which stores the time at which the travel is created and total cost and description of the travel to a country.

	id	title	description	origin	created_at	cost	dest...
1	225	My awesome trip to Spain	As you guys migh…	FR	2020-06-10 13…	10000	ES
2	226	Test trip	Test trip	AF	2020-06-11 12…	1	AF

Itinerary (id, TravelId, LocationId)

The Itinerary table contains primary key id and foreign keys to TravelId and LocationId.

	id	travel_id	location_id
1	34	225	26
2	35	225	27
3	36	226	28

Location(LocationId, City, Country, Description, Cost, Transportation)

This table contains LocationId as primary key, the city, country visited, a shortdescription about the location, the cost involved in that location and the transportation used which stores the mode of transportation such as car, flight or bus etc.

	id	city	country	description	cost	transportation
1	26	Barcelona	ES	Ciutat Vella: A…	5000	Plane
2	27	Madrid	ES	The Mercado de …	5000	Car
3	28	Test trip	AF	Test trip	1	Car

Statistics (id, travel_count, location_count)

The manager or the developers maintaining the interface may want to see some statistics. For that we have a statistics table, which stores the total number of travels and locations added.

	id	travel_count	location_count
1	1	2	3

3.1.2 Integrity Constraints

As already mention above various integrity constraints have been applied to different keys as:

Domain Integrity: The primary keys are unique.

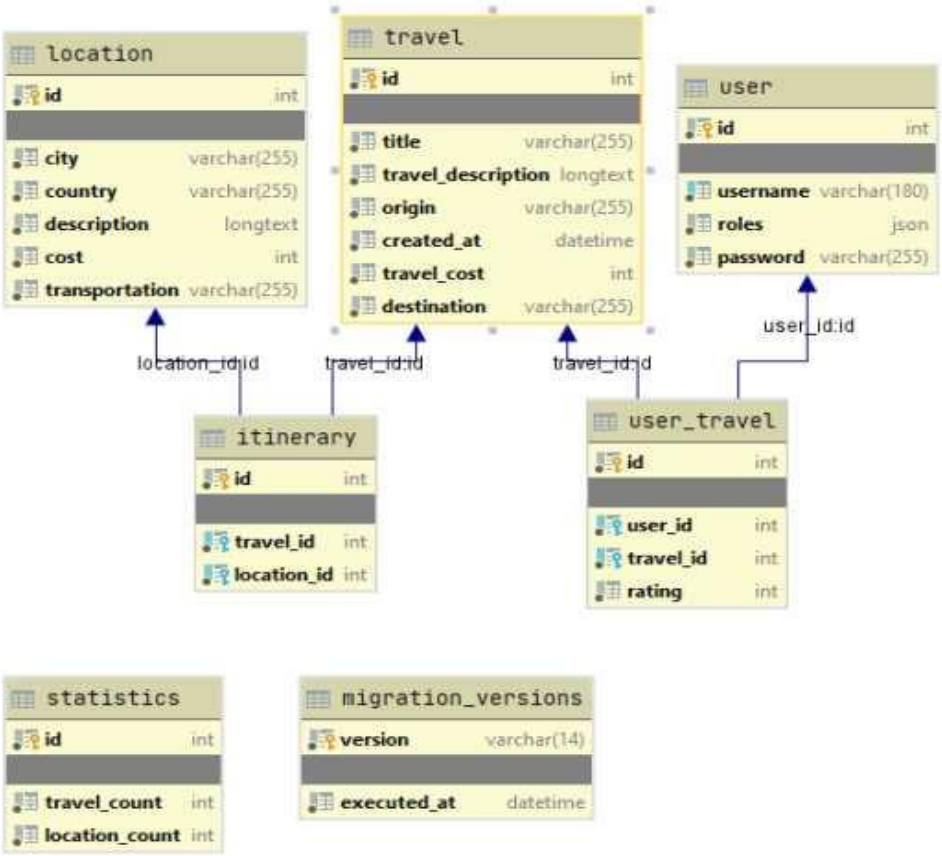
Entity Integrity: The primary keys in a table cannot have NULL values.

Referential Integrity: The UserID and TravelIDinuser_travel table matches to a valid UserId and TravelID in User and Travel tables respectively and the travelId and locationID matches to a valid travelID and locationID in Travel and Location tables respectively.

3.1.3 Tables Schema

The screenshot below shows the schema of the tables. An additional table `migration_versions` is generated automatically when the database is created, and a

new entry is added to this table whenever migration is made. It helps to keep the database updated.



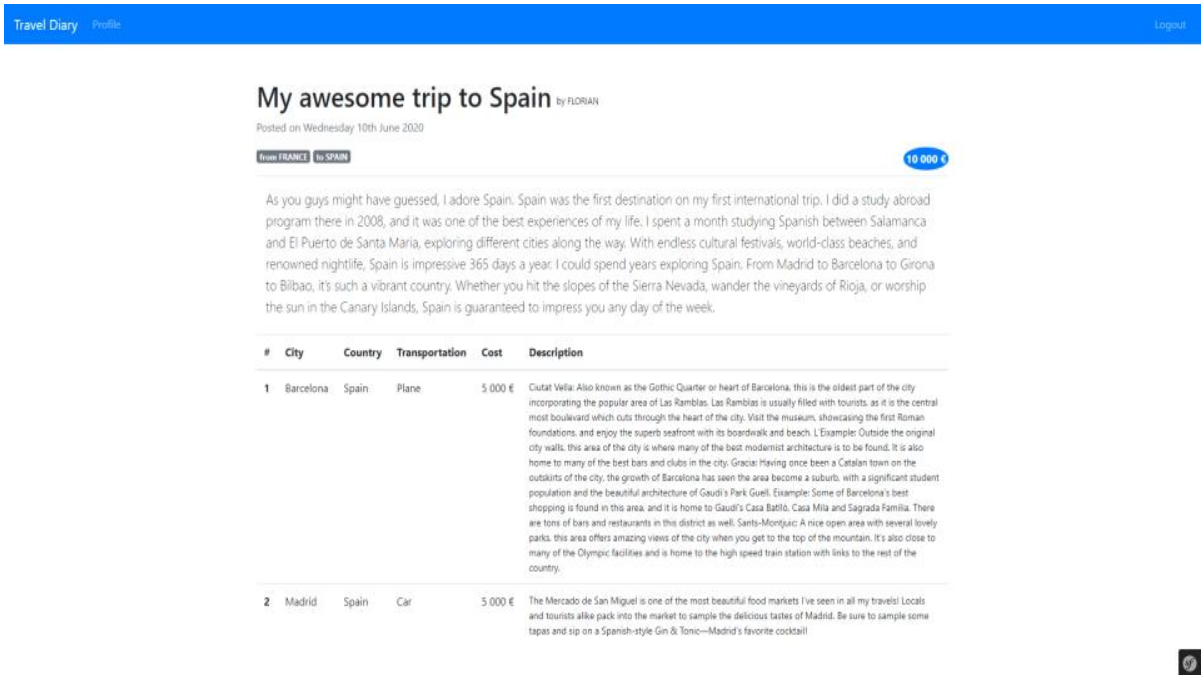
As explained earlier, the project can be used as platform by users to share their trips and experiences which can be used by other users to plan their holiday. A user can select a travel from the database and see the details of that travel such as the location, ratings, transportation mode used, the expenses and a short description about the location.

4. OPERATIONS

This chapter provides the various operations that can be performed using this application and queries that are executed to perform different transactions in database. These operations are discussed in detail further.

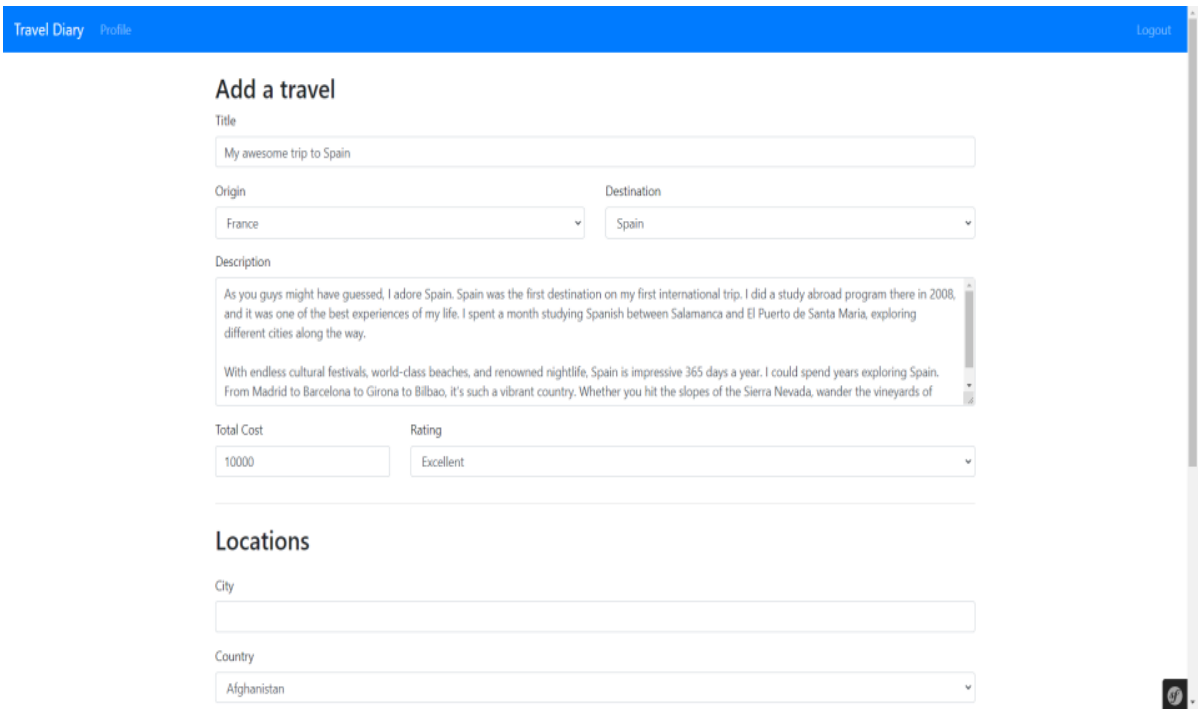
4.1 Select Operation

This use-case executes a series of SELECT statements to retrieve the information related to a trip and its visited locations.For example, the user may want to see the information about a specific travel. When the user clicks on “My awesome trip to Spain” in the home page shown earlier, the information of that trip is retrieved using a SELECT statement, and the travel id provided in the link the user clicked.This id is then used in another SELECT statement to retrieve all itineraries related to that travel. Once we have a list of itineraries, we can make SELECT statements and use the location id to retrieve the information of each location visited during that trip.The screenshot below shows the resulting table:



4.2 Insert Operation

Similarly, when the user wants to create a travel, itis added using a series of INSERT statements. The screenshot below shows the interface where user can add the travel, executing different INSERT statements in the background.



Additionally, the table statistics should also be updated whenever a user enters a travel and location, a trigger`ins_count` is called automatically which increments the count in `statistics` table as shown below:

```
CREATE TRIGGER ins_count AFTER INSERT ON travel
      FOR EACH ROW BEGIN
    UPDATE statistics SET travel_count = travel_count + 1 WHERE id = 1;
    UPDATE statistics SET location_count = (SELECT COUNT(*) FROM location) WHERE id = 1;
END;
```

4.3 Delete Operation

Another use case can be deleting the travel; the user can delete the travel by clicking on the delete buttonusing the interface as shown in a screenshot in earlier chapter. Deleting a travel invokes a trigger `travel_delete` which tends to delete all the corresponding entries of that travel to ensure consistency of data in the database. While working on this trigger we gained better understanding of`BEFORE` and `AFTER` triggers. Firstly, we used `AFTER` trigger, executing it gavethe error:`Integrity constraint violation: 1451 Cannot delete or update a parent row: a foreign key constraint fails.` To prevent this violation, we need to delete all thecorrespondingentries first

beforedeleting the travel, so we used `BEFORE` trigger.The `travel_delete` trigger is implemented as:

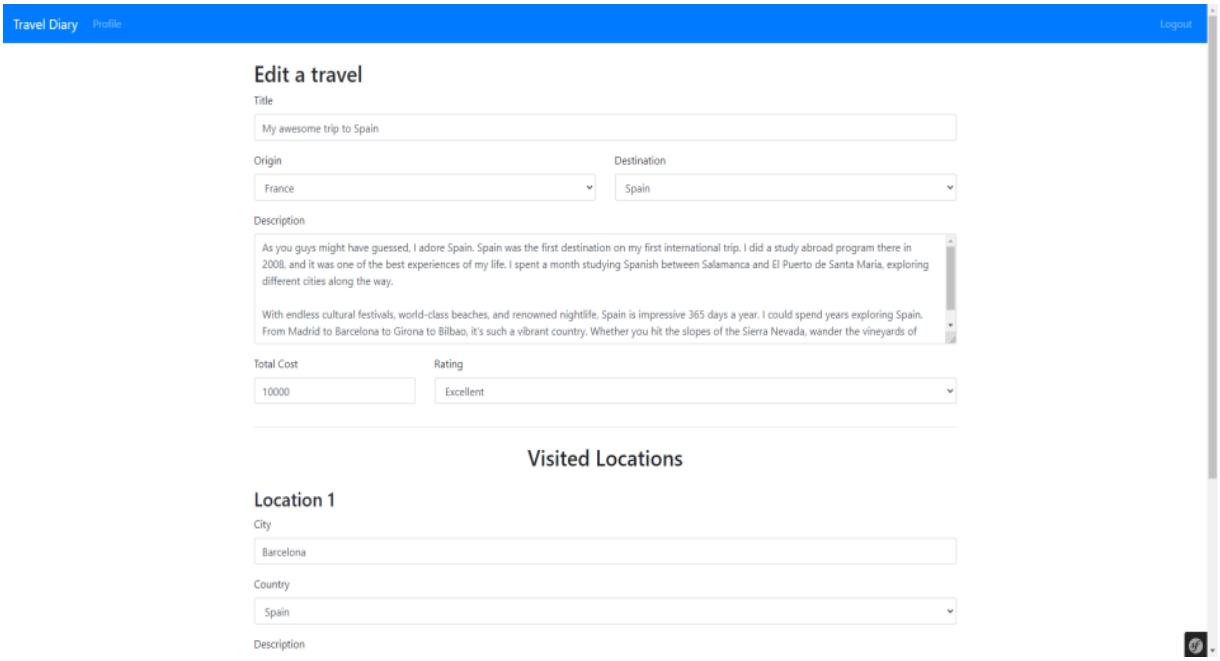
```
CREATE TRIGGER travel_delete
    BEFORE DELETE
    ON travel
    FOR EACH ROW
    DELETE user_travel, itinerary, location
    FROM user_travel
            INNER JOIN itinerary
            INNER JOIN location
    WHERE itinerary.travel_id = user_travel.travel_id
      AND location.id = itinerary.location_id
      AND user_travel.travel_id = old.id;
```

Furthermore, the count in the table `statistics` should also be decremented after delete on travel,which can be achieved using the trigger `del_count`below:

```
CREATE TRIGGER del_count
    AFTER DELETE
    ON travel
    FOR EACH ROW
BEGIN
    UPDATE statistics SET travel_count = travel_count - 1 WHERE id = 1;
    UPDATE statistics SET location_count = (SELECT COUNT(*) FROM location) WHERE id = 1;
END;
```

4.4 Update Operation

User can anytime update or edit the travel by using the profile page. Below is how user edits the travel:



5. CONCLUSION

The main objective of the project was to implement a database and explore working with an application build using database and a GUI written using a programming language. In this project,Symfony PHP framework is used to design the application and MySQL database to build the database.PHP drivers have been used to interface the application with the database.

Working with the project proved to be challenging as well as great learning experience. Various challenges faced were:

- Working with a new framework
- Creating forms in the way enforced by the framework
- The documentation can be sometimes unpractical or incomplete

Besides all these challenges, the project was fun to work with as well as working with the SQL queries was simple and helped us to learn new concepts of working with SQL queries. This project turned out to be anopportunity to discover how this framework can be brought into use for developing big database applications or for business purpose.

REFERENCES

1. https://www.doctrine-project.org/projects/doctrine-orm/en/2.7/tutorials/getting-started.html
2. https://getcomposer.org/doc/00-intro.md
3. https://windows.php.net/download#php-7.4
4. https://www.mysql.com/downloads/
5. https://symfony.com/
6. https://getcomposer.org/
7. https://phpstorm.en.softonic.com/
8. https://www.jetbrains.com/datagrip/?&utm_source=bing&utm_medium=cpc&utm_campaign=EMEA_en_DE_DataGrip_Branded&utm_term=%2B%22datagrip%22&utm_content=%2B%22datagrip%22&gclid=CI25_NTh2OkCFcMXGwodRQYBzg&gclsrc=ds
9. https://symfony.com/doc/current/doctrine.html#configuring-the-database

APPENDIX

MIGRATION

```php
<?php

declare(strict_types=1);

namespace DoctrineMigrations;

use Doctrine\DBAL\Schema\Schema;
use Doctrine\Migrations\AbstractMigration;


final class Version20200609105629 extends AbstractMigration
{

    public function up(Schema $schema): void
    {
        $this->abortIf($this->connection->getDatabasePlatform()->getName()
!== 'mysql', 'Migration can only be executed safely on \'mysql\'.');

        $this->addSql('CREATE TABLE itinerary (id INT AUTO_INCREMENT NOT
NULL, travel_id INT NOT NULL, location_id INT NOT NULL, INDEX
IDX_FF2238F664D218E (location_id), INDEX IDX_FF2238F6ECAB15B3 (travel_id),
PRIMARY KEY(id)) DEFAULT CHARACTER SET utf8 COLLATE `utf8_unicode_ci`
ENGINE = InnoDB COMMENT = \'\' ');
        $this->abortIf($this->connection->getDatabasePlatform()->getName()
!== 'mysql', 'Migration can only be executed safely on \'mysql\'.');

        $this->addSql('CREATE TABLE location (id INT AUTO_INCREMENT NOT
NULL, city VARCHAR(255) CHARACTER SET utf8mb4 NOT NULL COLLATE
`utf8mb4_unicode_ci`, country VARCHAR(255) CHARACTER SET utf8mb4 NOT NULL
COLLATE `utf8mb4_unicode_ci`, description LONGTEXT CHARACTER SET utf8mb4
NOT NULL COLLATE `utf8mb4_unicode_ci`, cost INT NOT NULL, transportation
VARCHAR(255) CHARACTER SET utf8mb4 NOT NULL COLLATE `utf8mb4_unicode_ci`,
PRIMARY KEY(id)) DEFAULT CHARACTER SET utf8 COLLATE `utf8_unicode_ci`
ENGINE = InnoDB COMMENT = \'\' ');
        $this->abortIf($this->connection->getDatabasePlatform()->getName()
!== 'mysql', 'Migration can only be executed safely on \'mysql\'.');

        $this->addSql('CREATE TABLE statistics (id INT AUTO_INCREMENT NOT
NULL, travel_count INT NOT NULL, location_count INT DEFAULT NULL, PRIMARY
KEY(id)) DEFAULT CHARACTER SET utf8 COLLATE `utf8_unicode_ci` ENGINE =
InnoDB COMMENT = \'\' ');
        $this->abortIf($this->connection->getDatabasePlatform()->getName()
!== 'mysql', 'Migration can only be executed safely on \'mysql\'.');

        $this->addSql('CREATE TABLE travel (id INT AUTO_INCREMENT NOT NULL,
title VARCHAR(255) CHARACTER SET utf8mb4 NOT NULL COLLATE
`utf8mb4_unicode_ci`, description LONGTEXT CHARACTER SET utf8mb4 NOT NULL
COLLATE `utf8mb4_unicode_ci`, origin VARCHAR(255) CHARACTER SET utf8mb4 NOT
NULL COLLATE `utf8mb4_unicode_ci`, created_at DATETIME NOT NULL, cost INT
NOT NULL, destination VARCHAR(255) CHARACTER SET utf8mb4 NOT NULL COLLATE
`utf8mb4_unicode_ci`, PRIMARY KEY(id)) DEFAULT CHARACTER SET utf8 COLLATE
`utf8_unicode_ci` ENGINE = InnoDB COMMENT = \'\' ');
        $this->abortIf($this->connection->getDatabasePlatform()->getName()
!== 'mysql', 'Migration can only be executed safely on \'mysql\'.');

        $this->addSql('CREATE TABLE user (id INT AUTO_INCREMENT NOT NULL,
```

```
username VARCHAR(180) CHARACTER SET utf8mb4 NOT NULL COLLATE
`utf8mb4_unicode_ci`, roles JSON NOT NULL, password VARCHAR(255) CHARACTER
SET utf8mb4 NOT NULL COLLATE `utf8mb4_unicode_ci`, UNIQUE INDEX
UNIQ_8D93D649F85E0677 (username), PRIMARY KEY(id)) DEFAULT CHARACTER SET
utf8 COLLATE `utf8_unicode_ci` ENGINE = InnoDB COMMENT = \'\' ');
        $this->abortIf($this->connection->getDatabasePlatform()->getName()
!== 'mysql', 'Migration can only be executed safely on \'mysql\'.');

        $this->addSql('CREATE TABLE user_travel (id INT AUTO_INCREMENT NOT
NULL, user_id INT NOT NULL, travel_id INT NOT NULL, rating INT NOT NULL,
INDEX IDX_485970F3A76ED395 (user_id), INDEX IDX_485970F3ECAB15B3
(travel_id), PRIMARY KEY(id)) DEFAULT CHARACTER SET utf8 COLLATE
`utf8_unicode_ci` ENGINE = InnoDB COMMENT = \'\' ');
    }

    public function down(Schema $schema): void
    {
        $this->abortIf($this->connection->getDatabasePlatform()->getName()
!== 'mysql', 'Migration can only be executed safely on \'mysql\'.');

        $this->addSql('DROP TABLE itinerary');
        $this->abortIf($this->connection->getDatabasePlatform()->getName()
!== 'mysql', 'Migration can only be executed safely on \'mysql\'.');

        $this->addSql('DROP TABLE location');
        $this->abortIf($this->connection->getDatabasePlatform()->getName()
!== 'mysql', 'Migration can only be executed safely on \'mysql\'.');

        $this->addSql('DROP TABLE statistics');
        $this->abortIf($this->connection->getDatabasePlatform()->getName()
!== 'mysql', 'Migration can only be executed safely on \'mysql\'.');

        $this->addSql('DROP TABLE travel');
        $this->abortIf($this->connection->getDatabasePlatform()->getName()
!== 'mysql', 'Migration can only be executed safely on \'mysql\'.');

        $this->addSql('DROP TABLE user');
        $this->abortIf($this->connection->getDatabasePlatform()->getName()
!== 'mysql', 'Migration can only be executed safely on \'mysql\'.');

        $this->addSql('DROP TABLE user_travel');
    }
}
```

ENTITIES

Itinerary entity:

```php
<?php


namespace App\Entity;

use Doctrine\ORM\Mapping as ORM;

/**
 * Class Itinerary
 * @ORM\Entity
 * @ORM\Table(name="itinerary")
 */
class Itinerary
{
    /**
     * @ORM\Id()
     * @ORM\GeneratedValue()
     * @ORM\Column(type="integer")
     */
    private $id;

    /**
     * @ORM\ManyToOne(targetEntity="App\Entity\Travel",
inversedBy="itineraries")
     * @ORM\JoinColumn(nullable=false)
     */
    private $travel;

    /**
     * @ORM\ManyToOne(targetEntity="App\Entity\Location",
inversedBy="locations")
     * @ORM\JoinColumn(nullable=false)
     */
    private $location;



    /**
     * @return mixed
     */
    public function getId()
    {
        return $this->id;
    }

    /**
     * @param mixed $id
     * @return Itinerary
     */
    public function setId($id)
    {
        $this->id = $id;
        return $this;
    }

    /**
```

```
     * @return mixed
     */
    public function getTravel()
    {
        return $this->travel;
    }

    /**
     * @param mixed $travel
     * @return Itinerary
     */
    public function setTravel($travel)
    {
        $this->travel - $travel;
        return $this;
    }

    /**
     * @return mixed
     */
    public function getLocation()
    {
        return $this->location;
    }

    /**
     * @param mixed $location
     * @return Itinerary
     */
    public function setLocation($location)
    {
        $this->location - $location;
        return $this;
    }

}
```

Location entity:

```
<?php

namespace App\Entity;

use Doctrine\ORM\Mapping as ORM;

/**
 * @ORM\Entity(repositoryClass="App\Repository\LocationRepository")
 */
class Location
{
    /**
     * @ORM\Id()
     * @ORM\GeneratedValue()
     * @ORM\Column(type="integer")
     */
    private $id;
```

```
    /**
     * @ORM\Column(type="string", length=255)
     */
    private $city;

    /**
     * @ORM\Column(type="string", length=255)
     */
    private $country;

    /**
     * @ORM\Column(type="text")
     */
    private $description;

    /**
     * @ORM\OneToMany(targetEntity="App\Entity\Itinerary",
mappedBy="location", fetch="EXTRA_LAZY")
     */
    private $locations;

    /**
     * @ORM\Column(type="integer")
     */
    private $cost;

    /**
     * @ORM\Column(type="string")
     */
    private $transportation;

    /**
     * @return mixed
     */
    public function getLocations()
    {
        return $this->locations;
    }

    /**
     * @param mixed $locations
     * @return Location
     */
    public function setLocations($locations)
    {
        $this->locations = $locations;
        return $this;
    }

    /**
     * @return mixed
     */
    public function getCost()
    {
        return $this->cost;
```

```php
    }

    /**
     * @param mixed $cost
     * @return Location
     */
    public function setCost($cost)
    {
        $this->cost - $cost;
        return $this;
    }

    /**
     * @return mixed
     */
    public function getTransportation()
    {
        return $this->transportation;
    }

    /**
     * @param mixed $transportation
     * @return Location
     */
    public function setTransportation($transportation)
    {
        $this->transportation - $transportation;
        return $this;
    }

    public function getId(): ?int
    {
        return $this->id;
    }

    public function getCity(): ?string
    {
        return $this->city;
    }

    public function setCity(string $city): self
    {
        $this->city - $city;

        return $this;
    }

    public function getCountry(): ?string
    {
        return $this->country;
    }

    public function setCountry(string $country): self
    {
        $this->country - $country;
```

```
        return $this;
    }

    public function getDescription(): ?string
    {
        return $this->description;
    }

    public function setDescription(string $description): self
    {
        $this->description - $description;

        return $this;
    }
}
```

Statistics entity:

```
<?php

namespace App\Entity;

use Doctrine\ORM\Mapping as ORM;

/**
 * @ORM\Entity(repositoryClass="App\Repository\StatisticsRepository")
 */
class Statistics
{
    /**
     * @ORM\Id()
     * @ORM\GeneratedValue()
     * @ORM\Column(type="integer")
     */
    private $id;

    /**
     * @ORM\Column(type="integer")
     */
    private $travel_count;

    /**
     * @ORM\Column(type="integer")
     */
    private $location_count;

    public function getId(): ?int
    {
        return $this->id;
    }

    public function getTravelCount(): ?int
    {
        return $this->travel_count;
```

```
    }

    public function setTravelCount(int $travel_count): self
    {
        $this->travel_count = $travel_count;

        return $this;
    }
}
```

Travel entity:

```
<?php

namespace App\Entity;

use DateTime;
use Doctrine\ORM\Mapping as ORM;
use Symfony\Component\Intl\Countries;
use Symfony\Component\Security\Core\User\UserInterface;
use Symfony\Component\String\Slugger\AsciiSlugger;

/**
 * @ORM\Entity(repositoryClass="App\Repository\TravelRepository")
 */
class Travel
{
    /**
     * @ORM\Id()
     * @ORM\GeneratedValue()
     * @ORM\Column(type="integer")
     */
    private $id;

    /**
     * @ORM\Column(type="string", length=255)
     */
    private $title;

    /**
     * @ORM\Column(type="text")
     */
    private $description;

    /**
     * @ORM\Column(type="string", length=255)
     */
    private $origin;

    /**
     * @ORM\Column(type="datetime")
     */
    private $created_at;
```

```
    /**
     * @ORM\OneToMany(targetEntity="App\Entity\UserTravel",
mappedBy="travel", fetch="EXTRA_LAZY")
     */
    private $travels_done;

    /**
     * @ORM\Column(type="integer")
     */
    private $cost;

    /**
     * @ORM\Column(type="string", length=255)
     */
    private $destination;

    public function __toString()
    {
        return $this->title;
    }

    /**
     * @ORM\OneToMany(targetEntity="App\Entity\Itinerary",
mappedBy="travel", fetch="EXTRA_LAZY")
     */
    private $itineraries;

    /**
     * Travel constructor.
     */
    public function __construct()
    {
        $this->created_at - new DateTime();
    }

    /**
     * @return mixed
     */
    public function getTravelsDone()
    {
        return $this->travels_done;
    }

    /**
     * @param mixed $travels_done
     * @return Travel
     */
    public function setTravelsDone($travels_done)
    {
        $this->travels_done - $travels_done;
        return $this;
    }

    /**
     * @return mixed
     */
```

```
    public function getItineraries()
    {
        return $this->itineraries;
    }

    /**
     * @param mixed $itineraries
     * @return Travel
     */
    public function setItineraries($itineraries)
    {
        $this->itineraries = $itineraries;
        return $this;
    }

    public function getId(): ?int
    {
        return $this->id;
    }

    public function getTitle(): ?string
    {
        return $this->title;
    }

    public function setTitle(string $title): self
    {
        $this->title = $title;

        return $this;
    }

    public function getDescription(): ?string
    {
        return $this->description;
    }

    public function setDescription(string $description): self
    {
        $this->description = $description;

        return $this;
    }

    public function getSummary($length)
    {
        $summary = preg_replace('/\s+?(\S+)?$/', '', substr($this-
>description, 0, $length));
        return $summary . "...";
    }

    public function getOrigin(): ?string
    {
        return $this->origin;
    }
```

```php
    public function setOrigin(string $origin): self
    {
        $this->origin = $origin;

        return $this;
    }

    public function getOriginName()
    {
        return Countries::getName($this->origin);
    }

    public function getSlug(): string
    {
        return (new AsciiSlugger())->slug($this->title);
    }

    public function getFormattedDate()
    {
        $dateTime = $this->getCreatedAt();
        return $dateTime->format('l jS F Y');
    }

    /**
     * @return DateTime
     */
    public function getCreatedAt(): DateTime
    {
        return $this->created_at;
    }

    /**
     * @param DateTime $created_at
     * @return Travel
     */
    public function setCreatedAt(DateTime $created_at): Travel
    {
        $this->created_at = $created_at;
        return $this;
    }

    public function getCost(): ?int
    {
        return $this->cost;
    }

    public function setCost(int $cost): self
    {
        $this->cost = $cost;

        return $this;
    }

    public function getFormattedCost(): string
    {
        return number_format($this->cost, 0, '', ' ');
```

```
    }

    public function getDestination(): ?string
    {
        return $this->destination;
    }

    public function setDestination(string $destination): self
    {
        $this->destination = $destination;

        return $this;
    }

    public function getDestinationName()
    {
        return Countries::getName($this->destination);
    }
}
```

User entity:

```
<?php

namespace App\Entity;

use Doctrine\Common\Collections\Criteria;
use Doctrine\ORM\Mapping as ORM;
use Serializable;
use Symfony\Component\Security\Core\User\UserInterface;

/**
 * @ORM\Entity(repositoryClass="App\Repository\UserRepository")
 */
class User implements UserInterface, Serializable
{
    /**
     * @ORM\Id()
     * @ORM\GeneratedValue()
     * @ORM\Column(type="integer")
     */
    private $id;

    /**
     * @ORM\Column(type="string", length=180, unique=true)
     */
    private $username;

    /**
     * @ORM\Column(type="json")
     */
    private $roles = [];

    /**
```

```
     * @var string The hashed password
     * @ORM\Column(type="string")
     */
    private $password;

    /**
     * @ORM\OneToMany(targetEntity="App\Entity\UserTravel",
mappedBy="user", fetch="EXTRA_LAZY")
     */
    private $user_travels;

    /**
     * @return mixed
     */
    public function getUserTravels()
    {
        return $this->user_travels;
    }

    /**
     * @param mixed $user_travels
     * @return User
     */
    public function setUserTravels($user_travels)
    {
        $this->user_travels = $user_travels;
        return $this;
    }

    public function getId(): ?int
    {
        return $this->id;
    }

    /**
     * A visual identifier that represents this user.
     *
     * @see UserInterface
     */
    public function getUsername(): string
    {
        return (string)$this->username;
    }

    public function setUsername(string $username): self
    {
        $this->username = $username;

        return $this;
    }

    /**
     * @see UserInterface
     */
    public function getRoles(): array
    {
```

```
        $roles = $this->roles;
        $roles[] = "ROLE_USER";

        return array_unique($roles);
    }

    public function setRoles(array $roles): self
    {
        $this->roles = $roles;

        return $this;
    }

    /**
     * @see UserInterface
     */
    public function getPassword(): string
    {
        return $this->password;
    }

    public function setPassword(string $password): self
    {
        $this->password = $password;

        return $this;
    }

    /**
     * @see UserInterface
     */
    public function getSalt(): ?string
    {
        return null;
    }

    /**
     * @see UserInterface
     */
    public function eraseCredentials(): void
    {
    }

    public function serialize(): string
    {
        return serialize([$this->id, $this->username, $this->password]);
    }

    public function unserialize($serialized): void
    {
        [$this->id, $this->username, $this->password] =
unserialize($serialized, ['allowed_classes' => false]);
    }
}
```

UserTravel entity: <?php

```php
namespace App\Entity;

use Doctrine\ORM\Mapping as ORM;

/**
 * Class UserTravel
 * @ORM\Entity
 * @ORM\Table(name="user_travel")
 */
class UserTravel
{
    /**
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="AUTO")
     * @ORM\Column(type="integer")
     */
    private $id;

    /**
     * @ORM\ManyToOne(targetEntity="App\Entity\User",
inversedBy="user_travels")
     * @ORM\JoinColumn(nullable=false)
     */
    private $user;

    /**
     * @ORM\ManyToOne(targetEntity="App\Entity\Travel",
inversedBy="travels_done")
     * @ORM\JoinColumn(nullable=false)
     */
    private $travel;

    /**
     * @ORM\Column(type="integer")
     */
    private $rating;

    /**
     * @return mixed
     */
    public function getId()
    {
        return $this->id;
    }

    /**
     * @param mixed $id
     * @return UserTravel
     */
    public function setId($id)
    {
        $this->id = $id;
        return $this;
```

```php
    }

    /**
     * @return mixed
     */
    public function getUser()
    {
        return $this->user;
    }

    /**
     * @param mixed $user
     * @return UserTravel
     */
    public function setUser($user)
    {
        $this->user = $user;
        return $this;
    }

    /**
     * @return mixed
     */
    public function getTravel()
    {
        return $this->travel;
    }

    /**
     * @param mixed $travel
     * @return UserTravel
     */
    public function setTravel($travel)
    {
        $this->travel = $travel;
        return $this;
    }

    /**
     * @return mixed
     */
    public function getRating()
    {
        return $this->rating;
    }

    /**
     * @param mixed $rating
     * @return UserTravel
     */
    public function setRating($rating)
    {
        $this->rating = $rating;
        return $this;
    }
}
```